Parse Tree

PROGRAM

<p4program> := <declarationList>

<declarationList> := { <declaration> | ; /* empty declaration */ }*

<declaration> := <variableDeclaration> | <externDeclaration> | <actionDeclaration> |
  <typeDeclaration> | <parserDeclaration> | <controlDeclaration> | <instantiation> |
  <errorDeclaration> | <matchKindDeclaration> | <functionDeclaration>

<nonTypeName> := IDENTIFIER | **apply** | **key** | **actions** | **state** | **~~entries~~** | **type**

<name> := <nonTypeName> | TYPE_IDENTIFIER

<nonTableKwName> := IDENTIFIER | TYPE_IDENTIFIER | **apply** | **state** | **type**

<parameterList> := { <parameter> { , <parameter> }* }

<parameter> := { <direction> } <typeRef> <name> { = <expression> }

<direction> := **in** | **out** | **inout**

<packageTypeDeclaration> := **package** <name> ~~<optTypeParameters>~~ (<parameterList>)

<instantiation> := <typeRef> (<argumentList>) <name> ;

<constructorParameters> := { (<parameterList>) }

PARSER

<parserDeclaration> := <parserTypeDeclaration> <constructorParameters>
  { <parserLocalElements> <parserStates> }

<parserLocalElements> := { <parserLocalElement> }*

<parserLocalElement> := <variableDeclaration> | <instantiation> ~~| <valueSetDeclaration>~~

<parserTypeDeclaration> := **parser** <name> ~~<optTypeParameters>~~ (<parameterList>)

<parserStates> := <parserState> { <parserState> }*

<parserState> := **state** <name> { <parserStatements> <transitionStatement> }

<parserStatements> := { <parserStatement> }*

<parserStatement> := <assignmentOrMethodCallStatement> | <directApplication> | <parserBlockStatement> |
  <variableDeclaration> | <emptyStatement>

<parserBlockStatement> := { <parserStatements> }

<transitionStatement> := { **transition** <stateExpression> }

<stateExpression> := <name> ; | <selectExpression>

<selectExpression> := **select** (<expressionList>) { <selectCaseList> }

<selectCaseList> := { <selectCase> }*

<selectCase> := <keysetExpression> : <name> ;

<keysetExpression> := <tupleKeysetExpression> | <simpleKeysetExpression>

<tupleKeysetExpression> := (<simpleExpressionList>)

<simpleExpressionList> := <simpleKeysetExpression> { , <simpleKeysetExpression }*

<simpleKeysetExpression> := <expression> ~~{ (**mask** | **range**) <expression> }~~ | **default** | _

~~<valueSetDeclaration> := **valueset** < (<baseType> | <tupleType> | <typeName>) >~~
~~(<expression>) <name> ;~~

CONTROL

<controlDeclaration> := <controlTypeDeclaration> <constructorParameters>
{ <controlLocalDeclarations> **apply** <blockStatement> }

<controlTypeDeclaration> := **control** <name> ~~<optTypeParameters>~~ (<parameterList>)

<controlLocalDeclaration> := <variableDeclaration> | <actionDeclaration> | <tableDeclaration> |
<instantiation>

<controlLocalDeclarations> := { <controlLocalDeclaration }*

EXTERN

<externDeclaration> := **extern**
(<nonTypeName> ~~<optTypeParameters>~~ { <methodPrototypes> }) |
(<functionPrototype> ;)

<methodPrototypes> := { <methodPrototype> }*

<functionPrototype> := <typeOrVoid> <name> ~~<optTypeParameters>~~ (<parameterList>)

<methodPrototype> :=
(<functionPrototype> ;) |
(TYPE (<parameterList>) ;) /* constructor */

TYPES

<typeRef> := <baseType> | <namedType> | <tupleType>

<namedType> := <typeName> ~~| <specializedType>~~ | <headerStackType>

<typeName> := TYPE_IDENTIFIER

<tupleType> := **tuple** < <typeArgumentList> >

<headerStackType> := <typeName> [<expression>]

~~<specializedType> := <typeName> < <typeArgumentList> >~~

<baseType> :=
bool | **error** | **string** | **void** |
int { < <integerTypeSize> > } |
bit { < <integerTypeSize> > } |
varbit < <integerTypeSize> >

<integerTypeSize> := INTEGER ~~| (<expression>)~~

<typeOrVoid> := <typeRef> | **void** | IDENTIFIER /* type variable */

~~<optTypeParameters> := { < <typeParameterList> > }~~

~~<typeParameterList> := <name> { , <name> }*~~

<realTypeArg> := _ | <typeRef>

<typeArg> := _ | <typeRef> | <nonTypeName>

<realTypeArgumentList> := <realTypeArg> { , <realTypeArg> }*

<typeArgumentList> := { <typeArg> { , <typeArg> }* }

<typeDeclaration> := <derivedTypeDeclaration> | <typedefDeclaration> | (<parserTypeDeclaration> ;)
| (<controlTypeDeclaration> ;) | (<packageTypeDeclaration> ;)

<derivedTypeDeclaration> := <headerTypeDeclaration> | <headerUnionDeclaration> |
<structTypeDeclaration> | <enumDeclaration>

<headerTypeDeclaration> := **header** <name> { <structFieldList> }

<headerUnionDeclaration> := **header_union** <name> { <structFieldList> }

<structTypeDeclaration> := **struct** <name> { <structFieldList> }

<structFieldList> := { <structField> { , <structField> }* }

<structField> := <typeRef> <name> ;

<enumDeclaration> := **enum** { **bit** < INTEGER > } <name> { <specifiedIdentifierList> }

<errorDeclaration> := **error** { <identifierList> }

<matchKindDeclaration> := **match_kind** { <identifierList> }

<identifierList> := <name> { , <name> }*

<specifiedIdentifierList> := <specifiedIdentifier> { , <specifiedIdentifier> }*

<specifiedIdentifier> := <name> { = <expression> }

<typedefDeclaration> := (**typedef** ~~| **type**~~) (<typeRef> | <derivedTypeDeclaration>) <name> ';'

STATEMENTS

<assignmentOrMethodCallStatement> := <lvalue>
(~~{ < <typeArgumentList> > }~~ (<argumentList>) ;) |
(= <expression> ;)

<emptyStatement> := ;

<returnStatement> := **return** { expression } ;

<exitStatement> := **exit** ;

<conditionalStatement> := **if** (<expression>) <statement> { **else** <statement> }

/* To support direct invocation of a control or parser without instantiation. */
<directApplication> := <typeName> . **apply** (<argumentList>) ;

<statement> := <assignmentOrMethodCallStatement> | <directApplication> | <conditionalStatement> |
<emptyStatement> | <blockStatement> | <exitStatement> | <returnStatement> | <switchStatement>

<blockStatement> := { <statementOrDeclList> }

<statementOrDeclList> := { <statementOrDeclaration> }*

<switchStatement> := **switch** (<expression>) { <switchCases> }

<switchCases> := { <switchCase> }*

<switchCase> := <switchLabel> : { <blockStatement> }

<switchLabel> := <name> | **default**

<statementOrDeclaration> := <variableDeclaration> | <statement> | <instantiation>

TABLES

<tableDeclaration> := **table** <name> { <tablePropertyList> }

<tablePropertyList> := <tableProperty> { <tableProperty> }*

<tableProperty> :=
(**key** = { <keyElementList> }) |
(**actions** = { <actionList> }) |
~~(**const entries** = { <entriesList> }) |~~ /* immutable entries */
~~({ **const** } <nonTableKwName> = <expression> ;)~~

<keyElementList> := { <keyElement> }*

<keyElement> := <expression> : <name> ;

<actionList> := { <actionRef> ; }*

<actionRef> := <nonTypeName> { (<argumentList>) }

<entriesList> := <entry> { <entry> }*

<entry> := <keysetExpression> : <actionRef> ;

<actionDeclaration> := **action** <name> (<parameterList>) <blockStatement>

VARIABLES

<variableDeclaration> := ~~{ **const** }~~ <typeRef> <name> { = <expression> };

EXPRESSIONS

<functionDeclaration> := <functionPrototype> <blockStatement>

<argumentList> := { <argument> { , <argument> }* }

<argument> := <expression> | _

<expressionList> := { <expression> { , <expression> }* }

<lvalue> := <nonTypeName> {
(. <name>) | /* member selector */
([<indexExpression>]) /* array subscript */
}*

<expression> := <expressionPrimary> { <exprOperator> <expression> }*

<expressionPrimary> := <integer> | <boolean> | <string> |
(~~{ . }~~ <nonTypeName>) |
({ <expressionList> }) |
((<expression>)) |
((! | ~ | -) <expression>) | /* unary expression */
(TYPE_IDENTIFIER | **error**) | /* member selector, function call, constructor */
((<typeRef>) <expression>) /* cast */

<exprOperator> := <binaryOperator> |
(. <name>) | /* member selector */
([<indexExpression>]) | /* array subscript */
((<argumentList>)) | /* function call */
~~(< <realTypeArgumentList> >) |~~
(= <expression>) /* named argument */

<indexExpression> := <expression> { : <expression> }

<integer> := INTEGER

<boolean> := **true** | **false**

<string> := STRING

<binaryOperator> := * | / | + | - | <= | >= | < | > | != | == | || | && | | | & | << | >>

Syntax Tree

PROGRAM

<p4program> := <declarationList>$_{\text{decl\_list}}$

<declarationList> := { <declaration>$_{[0..n]}$ }*

<declaration> := (<variableDeclaration> | <externDeclaration> | <actionDeclaration> |
<functionDeclaration> | <parserDeclaration> | <parserTypeDeclaration> | <controlDeclaration> |
<controlTypeDeclaration> | <typeDeclaration> | <errorDeclaration> | <matchKindDeclaration> |
<instantiation>)$_{\text{decl}}$

<name> := STRING$_{\text{strname}}$

<parameterList> := { <parameter>$_{[0..n]}$ }*

<parameter> := DIRECTION$_{\text{direction}}$ <typeRef>$_{\text{type}}$ <name>$_{\text{name}}$ { <expression> }$_{\text{init\_expr}}$

<packageTypeDeclaration> := <name>$_{\text{name}}$ ~~{ <typeParameterList> }$_{\text{type\_params}}$~~ <parameterList>$_{\text{params}}$

<instantiation> := <typeRef>$_{\text{type}}$ <argumentList>$_{\text{args}}$ <name>$_{\text{name}}$

PARSER

<parserDeclaration> := <typeDeclaration>$_{\text{proto}}$ { <parameterList> }$_{\text{ctor\_params}}$
<parserLocalElements>$_{\text{local\_elements}}$ <parserStates>$_{\text{states}}$

<parserTypeDeclaration> := <name>$_{\text{name}}$ ~~{ <typeParameterList> }$_{\text{type\_params}}$~~ <parameterList>$_{\text{params}}$

<parserLocalElements> := { <parserLocalElement>$_{[0..n]}$ }*

<parserLocalElement> := (<variableDeclaration> | <instantiation>)$_{\text{element}}$

<parserStates> := { <parserState>$_{[0..n]}$ }+

<parserState> := <name>$_{\text{name}}$ <parserStatements>$_{\text{stmt\_list}}$ <transitionStatement>$_{\text{transition\_stmt}}$

<parserStatements> := { <parserStatement>$_{[0..n]>}$ }*

<parserStatement> := (<assignmentStatement> | <functionCall> | <directApplication> |
<parserBlockStatement> | <variableDeclaration>)$_{\text{stmt}}$

<parserBlockStatement> := <parserStatements>$_{\text{stmt\_list}}$

<transitionStatement> := <stateExpression>$_{\text{stmt}}$

<stateExpression> := (<name> | <selectExpression>)$_{\text{expr}}$

<selectExpression> := <expressionList>$_{\text{expr\_list}}$ <selectCaseList>$_{\text{case\_list}}$

<selectCaseList> := { <selectCase>$_{[0..n]}$ }*

<selectCase> := <keysetExpression>$_{\text{keyset\_expr}}$ <name>$_{\text{name}}$

<keysetExpression> := (<tupleKeysetExpression> | <simpleKeysetExpression>)$_{\text{expr}}$

<tupleKeysetExpression> := <simpleExpressionList>$_{\text{expr\_list}}$

<simpleKeysetExpression> := (<expression> | <default> | <dontcare>)$_{\text{expr}}$

<simpleExpressionList> := { <simpleKeysetExpression>$_{[0..n]}$ }+

CONTROL

<controlDeclaration> := <typeDeclaration>$_{\text{proto}}$ { <parameterList> }$_{\text{ctor\_params}}$
<controlLocalDeclarations>$_{\text{local\_decls}}$ <blockStatement>$_{\text{apply\_stmt}}$

<controlTypeDeclaration> := <name>$_{\text{name}}$ ~~{ <typeParameterList> }$_{\text{type\_params}}$~~ <parameterList>$_{\text{params}}$

<controlLocalDeclarations> := { <controlLocalDeclaration>$_{[0..n]}$ }*

<controlLocalDeclaration> := (<variableDeclaration> | <actionDeclaration> | <tableDeclaration> |
<instantiation>)$_{\text{decl}}$

EXTERN

<externDeclaration> := (<externTypeDeclaration> | <functionPrototype>)$_{\text{decl}}$

<externTypeDeclaration> := <name>$_{\text{name}}$ ~~{ <typeParameterList>$_{\text{type\_params}}$ }~~ <methodPrototypes>$_{\text{method\_protos}}$

<methodPrototypes> := { <functionPrototype>$_{[0..n]}$ }*

<functionPrototype> := { <typeRef> }$_{\text{return\_type}}$ <name>$_{\text{name}}$ ~~{ <typeParameterList> }$_{\text{type\_params}}$~~
<parameterList>$_{\text{params}}$

TYPES

<typeRef> := (<baseTypeBoolean> | <baseTypeInteger> | <baseTypeBit> | <baseTypeVarbit> |
<baseTypeString> | <baseTypeVoid> | <baseTypeError> | <name> | ~~<specializedType> |~~
<headerStackType> | <tupleType>)$_{\text{type}}$

<tupleType> := <typeArgumentList>$_{\text{type\_args}}$

<headerStackType> := <typeRef>$_{\text{type}}$ <expression>$_{\text{stack\_expr}}$

~~<specializedType> := <typeRef>$_{\text{type}}$ <typeArgumentList>$_{\text{type\_args}}$~~

<baseTypeBoolean> := <name>$_{\text{name}}$
<baseTypeInteger> := <name>$_{\text{name}}$ { <integerTypeSize> }$_{\text{size}}$
<baseTypeBit> := <name>$_{\text{name}}$ { <integerTypeSize> }$_{\text{size}}$
<baseTypeVarbit> := <name>$_{\text{name}}$ <integerTypeSize>$_{\text{size}}$
<baseTypeString> := <name>$_{\text{name}}$
<baseTypeVoid> := <name>$_{\text{name}}$
<baseTypeError> := <name>$_{\text{name}}$

<integerTypeSize> := INTEGER$_{\text{size}}$

~~<typeParameterList> := { <name>$_{[0..n]}$ }+~~

<realTypeArg> := (<typeRef> | <dontcare>)$_{\text{arg}}$

<typeArg> := (<typeRef> | <name> | <dontcare>)$_{\text{arg}}$

<realTypeArgumentList> := { <realTypeArg>$_{[0..n]}$ }+

<typeArgumentList> := { <typeArg>$_{[0..n]}$ }*

<typeDeclaration> := (<derivedTypeDeclaration> | <typedefDeclaration> | <parserTypeDeclaration> |
<controlTypeDeclaration> | <packageTypeDeclaration>)$_{\text{decl}}$

<derivedTypeDeclaration> := (<headerTypeDeclaration> | <headerUnionDeclaration> |
<structTypeDeclaration> | <enumDeclaration>)$_{\text{decl}}$

<headerTypeDeclaration> := <name>$_{\text{name}}$ <structFieldList>$_{\text{fields}}$

<headerUnionDeclaration> := <name>$_{\text{name}}$ <structFieldList>$_{\text{fields}}$

<structTypeDeclaration> := <name>$_{\text{name}}$ <structFieldList>$_{\text{fields}}$

<structFieldList> := { <structField>$_{[0..n]}$ }*

<structField> := <typeRef>$_{\text{type}}$ <name>$_{\text{name}}$

<enumDeclaration> := INTEGER$_{\text{type\_size}}$ <name>$_{\text{name}}$ <specifiedIdentifierList>$_{\text{fields}}$

<errorDeclaration> := <identifierList>$_{\text{fields}}$

<matchKindDeclaration> := <identifierList>$_{\text{fields}}$

<identifierList> := { <name>$_{[0..n]}$ }+

<specifiedIdentifierList> := { <specifiedIdentifier>$_{[0..n]}$ }+

<specifiedIdentifier> := <name>$_{\text{name}}$ { <expression> }$_{\text{init\_expr}}$

<typedefDeclaration> := (<typeRef> | <derivedTypeDeclaration>)$_{\text{type\_ref}}$ <name>$_{\text{name}}$

STATEMENTS

<assignmentStatement> := (<expression> | <lvalueExpression>)$_{\text{lhs\_expr}}$ <expression>$_{\text{rhs\_expr}}$

<functionCall> := (<expression> | <lvalueExpression>)$_{\text{lhs\_expr}}$ <argumentList>$_{\text{args}}$

<returnStatement> := { <expression> }$_{\text{expr}}$

<exitStatement> := **exit**

<conditionalStatement> := <expression>$_{\text{cond\_expr}}$ <statement>$_{\text{stmt}}$ { <statement>$_{\text{else\_stmt}}$ }

<directApplication> := (<name> | <typeRef>)$_{\text{name}}$ <argumentList>$_{\text{args}}$

<statement> := (<assignmentStatement> | <functionCall> | <directApplication> |
<conditionalStatement> | <emptyStatement> | <blockStatement> | <exitStatement> |
<returnStatement> | <switchStatement>)$_{\text{stmt}}$

<blockStatement> := <statementOrDeclList>$_{\text{stmt\_list}}$

<statementOrDeclList> := { <statementOrDeclaration>$_{[0..n]}$ }*

<switchStatement> := <expression>$_{\text{expr}}$ <switchCases>$_{\text{switch\_cases}}$

<switchCases> := { <switchCase>$_{[0..n]}$ }*

<switchCase> := <switchLabel>$_{\text{label}}$ { <blockStatement>$_{\text{stmt}}$ }

<switchLabel> := (<name> | <default>)$_{\text{label}}$

<statementOrDeclaration> := (<variableDeclaration> | <statement> | <instantiation>)$_{\text{stmt}}$

TABLES

<tableDeclaration> := <name>$_{\text{name}}$ <tablePropertyList>$_{\text{prop\_list}}$

<tablePropertyList> := { tableProperty$_{[0..n]}$ }+

<tableProperty> := (<keyProperty> | <actionsProperty> ~~| <entriesProperty> | <simpleProperty>~~)$_{\text{prop}}$

<keyProperty> := <keyElementList>$_{\text{keyelem\_list}}$

<keyElementList> := { <keyElement>$_{[0..n]}$ }*

<keyElement> := <expression>$_{\text{expr}}$ <name>$_{\text{match}}$

<actionsProperty> := <actionList>$_{\text{action\_list}}$

<actionList> := { <actionRef>$_{[0..n]}$ }*

<actionRef> := <name>$_{\text{name}}$ { <argumentList>$_{\text{args}}$ }

~~<entriesProperty> := <entriesList>$_{\text{entries\_list}}$~~

~~<entriesList> := { <entry>$_{[0..n]}$ }+~~

~~<entry> := <keysetExpression>$_{\text{keyset}}$ <actionRef>$_{\text{action}}$~~

~~<simpleProperty> := <name>$_{\text{name}}$ <expression>$_{\text{init\_expr}}$~~

<actionDeclaration> := <name>$_{\text{name}}$ <parameterList>$_{\text{params}}$ <blockStatement>$_{\text{stmt}}$

VARIABLES

<variableDeclaration> := <typeRef>$_{\text{type}}$ <name>$_{\text{name}}$ { <expression> }$_{\text{init\_expr}}$

EXPRESSIONS

<functionDeclaration> := <functionPrototype>$_{\text{proto}}$ <blockStatement>$_{\text{stmt}}$

<argumentList> := { <argument>$_{[0..n]}$ }*

<argument> := (<expression> |)$_{\text{arg}}$

<expressionList> := { <expression>$_{[0..n]}$ }*

<lvalueExpression> := (<name> | <memberSelector> | <arraySubscript>)$_{\text{expr}}$

<expression> := (<expression> | <booleanLiteral> | <integerLiteral> | <stringLiteral> | <name> |
<expressionList> | <castExpression> | <unaryExpression> | <binaryExpression> | <memberSelector> |
<arraySubscript> | <functionCall>)$_{\text{expr}}$ ~~{ <realTypeArgumentList> }$_{\text{type\_args}}$~~

<castExpression> := <typeRef>$_{\text{type}}$ <expression>$_{\text{expr}}$

<unaryExpression> := OPERATOR$_{\text{op}}$ <expression>$_{\text{operand}}$

<binaryExpression> := <expression>$_{\text{left\_operand}}$ OPERATOR$_{\text{op}}$ <expression>$_{\text{right\_operand}}$

<memberSelector> := (<expression> | <lvalueExpression>)$_{\text{lhs\_expr}}$ <name>$_{\text{name}}$

<arraySubscript> := (<expression> | <lvalueExpression>)$_{\text{lhs\_expr}}$ <indexExpression>$_{\text{index\_expr}}$

<indexExpression> := <expression>$_{\text{start\_index}}$ { <expression> }$_{\text{end\_index}}$

<booleanLiteral> := INTEGER$_{\text{value}}$

<integerLiteral> := INTEGER$_{\text{value}}$ INTEGER$_{\text{width}}$

<stringLiteral> := STRING$_{\text{value}}$

<default> := **default**

<dontcare> := _